ChinaXiv合作期刊

NUCLEAR SCIENCE AND TECHNIQUES **26**, 050502 (2015)

# Scalability of 3D deterministic particle transport on the Intel MIC architecture*

WANG Qing-Lin (王庆林),[1,†] LIU Jie (刘杰),[1] GONG Chun-Ye (龚春叶),[2] and XING Zuo-Cheng (邢座程)[1]

[1]*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China*

[2]*Science and Technology on Space Physics Laboratory, Beijing 100076, China*

(Received November 19, 2014; accepted in revised form January 5, 2015; published online October 20, 2015)

The key to large-scale parallel solutions of deterministic particle transport problem is single-node computation performance. Hence, single-node computation is often parallelized on multi-core or many-core computer architectures. However, the number of on-chip cores grows quickly with the scale-down of feature size in semiconductor technology. In this paper, we present a scalability investigation of one energy group time-independent deterministic discrete ordinates neutron transport in 3D Cartesian geometry (Sweep3D) on Intel's Many Integrated Core (MIC) architecture, which can provide up to 62 cores with four hardware threads per core now and will own up to 72 in the future. The parallel programming model, OpenMP, and vector intrinsic functions are used to exploit thread parallelism and vector parallelism for the discrete ordinates method, respectively. The results on a 57-core MIC coprocessor show that the implementation of Sweep3D on MIC has good scalability in performance. In addition, the application of the Roofline model to assess the implementation and performance comparison between MIC and Tesla K20C Graphics Processing Unit (GPU) are also reported.

Keywords: Particle transport, Discrete ordinates method, Sweep3D, Many Integrated Core (MIC), Scalability, Roofline model, Graphics Processing Unit (GPU)

DOI: 10.13538/j.1001-8042/nst.26.050502

chinaXiv:202306.00232v1

## I. INTRODUCTION

The discrete ordinates ($S_n$) method is one of the most popular deterministic numerical methods for solving particle transport equations [1]. The particle transport equation depends on the energy, angular directions and spatial coordinates of the particles with an underlying medium so that the solution is highly computation-intensive [2]. Some works [3–5] reveal that single-node computation speed is the key performance factor, rather than inter-processor communication performance, in the discrete ordinates method for large-scale problems. The advent of on-chip multi-core or many-core computer architectures makes it possible to overcome the challenges above.

The Sweep3D benchmark [6] is a time-independent, single-group, discrete ordinates 3D Cartesian geometry deterministic neutron transport code, which is extracted from real Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI) applications. It represents the heart of the discrete ordinates method for solving particle transport equations. Recently, Sweep3D has been accelerated on different multi-core or many-core computer architectures [7–13]. Petrini *et al*. [7] ported Sweep3D to the Cell Broadband Engine (B.E) processor with an on-chip single-MPI routine and the implementation acquired good performance by means of exploiting different levels of parallelism and data streaming in Sweep3D. In contrast with Petrini *et al*., Lubeck *et al*. [8] developed a library of on-chip SPU-to-SPU communication routines and utilized on-chip multi-MPI routines to accomplish an SPU-centric implementation of Sweep3D on Cell B.E.. The latter got better performance owing to the decrease of data movement. With the application of many-core architectures in high performance computing, Sweep3D has been implemented on Graphics Processing Unit (GPU) [9, 10] and Intel Many Integrated Core (MIC) [13]. Gong *et al*. [9] used multi-dimensional optimizations to parallelize Sweep3D on GPU and a speedup of 2.25 times was reached in comparison to the CPU-based version when flux fixup was disabled. Moreover, more concurrent threads were obtained by exacting parallelism in the data-dependent loop of solving recursive $S_n$ equations in Sweep3D and then further performance improvement on GPU was achieved [10]. Wang *et al*. [13] applied both hardware threads and vector units in MIC to efficiently exploit multi-level parallelism in Sweep3D and got a 1.99 times speedup based on an eight-core Intel Xeon E5-2660 CPU.

The number of transistors integrated on a single chip increases exponentially as the semiconductor industry continues to scale down feature size. The growth in the number of transistors permits the rapid development of Chip Multiprocessor (CMP) architectures. One evolving trend of CMP architectures is integrating more and more general-purpose cores on each chip, such as the Intel MIC architecture [14] which is built on light-weight x86 cores. MIC was introduced as a highly parallel coprocessor by Intel Corporation in 2010. The earlier prototype card of the Intel MIC architecture, codenamed Knights Ferry (KNF), provides up to 32 cores with four threads per core in a single chip. The first generation product, codenamed Knights Corner (KNC), consists of up to 62 cores per chip. The second generation product, codenamed Knights Landing (KNL), will feature up to 72 cores with four

* Supported by National Natural Science Foundation of China (Nos. 61402039, 61170083, 60970033, 61373032 and 91430218), National High Technology Research and Development Program of China (No. 2012AA01A301), China Postdoctoral Science Foundation (No. 2014M562570) and National Key Basic Research Program of China (No. 61312701001)

† Corresponding author, wangqinglin.thu@gmail.com

ChinaXiv合作期刊



threads per core in future [15]. Therefore, it's important to study how parallel algorithms scale with the number of cores on CMP architectures. Wang *et al*. [16] studied the scalability of the embarrassingly parallel algorithm on MIC. Saule *et al*. [17] evaluated the scalability of three graph algorithms on MIC. However, there is little work about the scalability evaluation of 3D deterministic particle transport on the Intel MIC architecture.

The work in this paper is based on our previous work about the parallelization of 3D deterministic particle transport on MIC [13]. We present the investigation of the scalability of 3D deterministic particle transport algorithms on MIC. Our results on an MIC chip, including 57 cores, show that the algorithm can scale well with the rapidly increasing number of cores in MIC. Without flux fixup, the performance of the algorithm is determined by the number of cores and the off-chip memory bandwidth in the MIC. With flux fixup, most of the algorithm cannot be vectorized, then the performance of the algorithm is mainly dependent on the number of cores in the MIC. Additionally, we apply the Roofline model to assess the implementation of Sweep3D on MIC, and also compare it with the GPU implementation running on Tesla K20C GPU.

The structure of this paper is as follows. Section II presents the relative background. Section III describes MIC-based Sweep3D implementation in detail. The results of performance and scalability are collected in Sec. IV. In Sec. V, we conclude and describe future work.

## II. BACKGROUND

### A. Sweep3D

The time-independent single-group particle transport equation is shown in Eq. (1). The unknown field is $\Psi(\vec{r},\vec{\Omega})$, which represents the flux of particles traveling in direction $\vec{\Omega}$ at spatial point $\vec{r}$. $\sigma_t$ is the total cross section and $\sigma_s$ is the scattering cross section from $\vec{\Omega}'$ into $\vec{\Omega}$ at $\vec{r}$. The right-hand side of the equation is the source item, including the scattering source and external source. $Q_{\text{ext}}(\vec{r},\vec{\Omega})$ expresses the external source.

$$\vec{\Omega}\cdot\nabla\Psi(\vec{r},\vec{\Omega})+\sigma_t(\vec{r})\Psi(\vec{r},\vec{\Omega})=\int_{4\pi}\sigma_s(\vec{r},\vec{\Omega}'\rightarrow\vec{\Omega})\Psi(\vec{r},\vec{\Omega}')d\vec{\Omega}'+Q_{\text{ext}}(\vec{r},\vec{\Omega}). \quad (1)$$

In Sweep3D, the angular-direction, $\Psi$, is discretized into a set of quadrature points and the space is divided into a finite mesh of cells. The XYZ geometry is represented by an IJK logically rectangular grid of cells. Source Iteration (SI) scheme is used to deal with the angular couplings by scattering media. Each iteration includes computing the iterative source, wavefront sweeping, computing flux error, and judging whether the convergence condition is met or not. Wavefront sweeping is the most time-consuming part. In Cartesian geometry, each octant of angle sweeps has a different sweep direction through the physical space, and all angles in a given octant sweep the same way. Integrating the left-hand and right-hand sides of Eq. (1) over the neighboring angular-directions region, $\Delta\vec{\Omega}_m$, of a given discrete angle, $\vec{\Omega}_m(\mu_m,\eta_m,\xi_m)$, we get the balanced equation as follows:

$$\mu_m\frac{\partial\Psi_m}{\partial x}+\eta_m\frac{\partial\Psi_m}{\partial y}+\xi_m\frac{\partial\Psi_m}{\partial z}+\sigma_t(\vec{r})\Psi_m=Q_m(\vec{r}). \quad (2)$$

In SI scheme, $Q_m(\vec{r})$ is known. The diamond space difference scheme is applied to get three auxiliary equations so that each grid cell has 4 equations (3 auxiliary plus 1 balance) with 7 unknowns (6 faces plus 1 central). Boundary conditions initialize the sweep and allow the system of equations to be completed. For any given cell, three known inflows make the cell center and three outflows can be obtained, and then the three outflows provide inflows to three adjoining cells in particle traveling directions. Therefore, there is a recursion dependence in all three grid directions. The recursion dependence causes the sweep to proceed in a diagonal wave across the physical space, shown in Fig. 1. Therefore, the parallelism is limited by the length of the JK-diagonal line. To alleviate this problem, MMI angles for each octant are pipelined on JK-diagonal lines to increase the number of parallel I-lines.

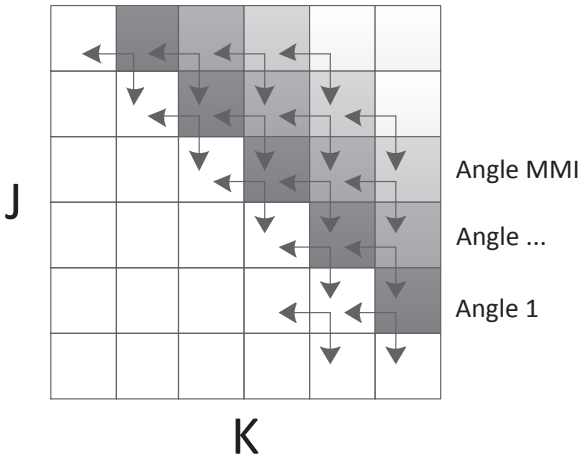


Fig. 1. Wavefront sweeping and angle pipelining in Sweep3D.

Moreover, Sweep3D utilizes Diffusion Synthetic Acceleration (DSA) [18] to improve its convergence of source iteration scheme. So wavefront sweeping subroutine mainly involves computing sources from spherical harmonic ($P_n$) moments, solving $S_n$ equation recursively with or without flux fixup, updating flux from $P_n$ moments, and updating DSA face currents, as shown in Fig. 2.



chinaXiv:202306.00232v1

ChinaXiv合作期刊



```
1  for iq = 1 to 8 do                                          // octants
2      for mo = 1 to mmo do                                    // angle pipelining loop
3          for kk = 1 to kb do                                 // k-plane pipeling loop
4              RECV East/West                                  // recv block I-inflows
5              RECV North/South                                // recv block J-inflows
6              for idiag = 1 to jt + nk − 1 + mmi − 1 do
               // JK-diagonals with MMI pipelining
7                  for jkm = 1 to ndiag do // I-line grid columns
8                      for i = 1 to it do
9                          Compute source from P_n moments
10                     if not do fixup then
11                         for i = 1 to it do
12                             Solve S_n equation
13                     else
14                         for i = 1 to it do
15                             Solve S_n equation with fixup
16                     for i = 1 to it do
17                         Update flux from P_n moments
18                     for i = 1 to it do
19                         Update DSA face currents
20             SEND East/West                                  // send block I-inflows
21             SEND North/South                                // send block J-inflows
```

Fig. 2. Wavefront sweeping subroutine in Sweep3D.

### B. MIC Architecture

The MIC architecture is shown in Fig. 3. Each MIC chip consists of many general-purpose cores. These cores are in-order and issue two instructions per cycle to two asymmetrical pipelines (u and v-pipes). The two pipelines can both deal with scalar operations, whereas only the u-pipe can execute vector operations. One vector unit in MIC can perform 16 single precision or 8 double precision floating point operations simultaneously. The vector units also support the floating point fused multiply-add operations. Each core maintains four hardware threads and schedules these threads by round robin. Each core also owns a 32-KB private L1 instruction cache, 32-KB private L1 data cache, and 512-KB unified L2 cache. The two-level caches of all cores are kept coherent by a distributed tag directory. There are multi on-die Memory Controllers (MCs), which are used to connect off-chip high-bandwidth Graphics Double-Data Rate, version 5 memory (GDDR5) to on-chip cores and L2 caches in the MIC. All cores, MCs, and caches on the chip communicate with each other by a high-bandwidth bidirectional ring bus.

MIC architecture supports many parallel programming models, such as Open Multi-Processing (OpenMP), Intel Cilk Plus, and Intel Threading Building Blocks (TBB). In the following implementation, we use OpenMP to exploit the computing capability of parallel hardware threads in MIC. The ways to make use of the vector units in MIC include compiler automatic vectorization and vector intrinsics functions. As compiler automatic vectorization is only possible for simple loops, we apply intrinsics functions to exploit the vector parallelism in the discrete ordinates method. Additionally, two kinds of usage models are available in MIC, offload and native. In the offload model, MIC runs as a coprocessor like

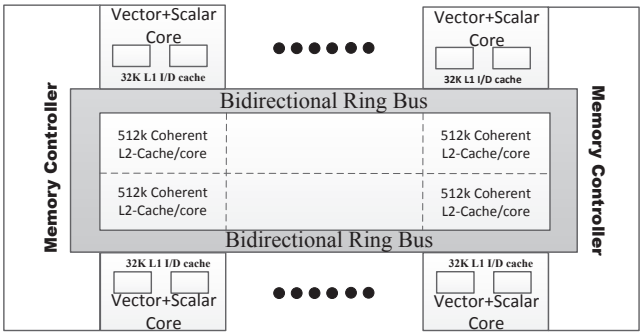


Fig. 3. MIC architecture.

GPU. In the native model, MIC acts as a many-core processor. In contrast with the usage model of GPU, the native model is unique to MIC. In order to eliminate the influence of communication between the CPU host and MIC coprocessor, we use the native model of MIC to evaluate the scalability of 3D deterministic particle transport on MIC.

## III. DETAILS OF MIC-BASED IMPLEMENTATION

To achieve full utilization of parallel computing resources in MIC, one MIC-based particle transport solver is proposed in Fig. 4. In the solver, we use the mechanism of thread parallelism and vector parallelism to exploit the parallelism in each procedure of the discrete ordinate method.

### A. Thread Parallelism

As shown in Fig. 4, all three procedures of Sweep3D are processed with many parallel threads, while thread parallelism in these procedures is different.

The iterative source is equal to iterative scattering moments plus the external source, shown in Eq. (3):

$$S_l(\vec{r})_i = \sigma_s(\vec{r})\Phi_l(\vec{r})_{i-1} + [Q_{\text{ext}}(\vec{r})]_{l==0}, \qquad (3)$$

where $i$ presents the $i^{\text{th}}$ iteration loop. There is no dependence among the calculations of iterative source of all cells so that the iterative source of all cells in each iteration can be computed in parallel. The OpenMP primitive *parallel_for* can be utilized to schedule the parallel tasks to all parallel hardware threads. The granularity of scheduling the tasks can be one I-line grid column and one k-plane grid block. In Sweep3D, the number of k-plane grid blocks may be too small to make the tasks occupy all the hardware threads. Hence, one I-line grid column is set to the minimum granularity of partitioning the tasks in the thread-level parallelization of the computing iterative source.

As one JK-diagonal line provides the input to the adjoining JK-diagonal line in the particle traveling direction, it's very difficult to extract thread-level parallelism in wavefront-sweeping. Fortunately, all I-lines in each JK-diagonal line are independent of each other. The parallelism is the number of

chinaXiv:202306.00232v1

ChinaXiv合作期刊



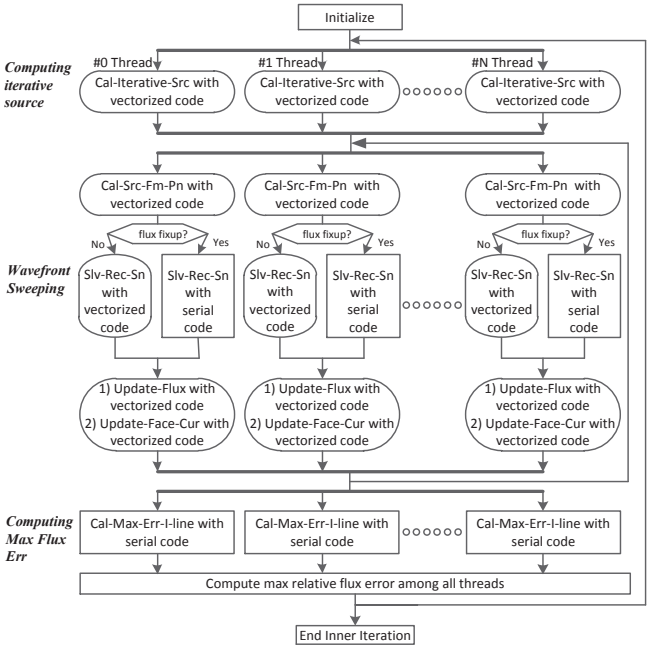


Fig. 4. Work flow of the MIC-based particle transport solver. Cal-Iterative-Src: compute source from external source plus scattering moments. Cal-Src-Fm-Pn: computing source from $P_n$ Moments. Slv-Rec-Sn: solving recursive $S_n$ equations. Update-Flux: updating flux from $P_n$ moments. Update-Face-Cur: updating DSA face currents. Cal-Max-Err-I-line: computing max relative flux error among cells in I-line grid columns.

I-lines in JK-diagonal lines and varies with particle transporting. The minimum number of I-lines in all JK-diagonal lines is one and the maximum is the problem size of the J or K dimensions in Fig. 1. There are a large number of parallel hardware threads in MIC, and it's important to increase the amount of parallelism. Therefore, the *MMI* angles for each octant are pipelined. As shown in Fig. 1, the problem sizes of the I and K dimension are both six, and three different angles are processed in pipeline. The depicted *idiag* value is five, corresponding to Line 6 in Fig. 2. The number of parallel I-lines for the *idiag* value is only five without pipelining angles and that arises to twelve with the pipelining of three angles. The OpenMP primitive *parallel_for* is added to parallel all the I-lines of all JK-diagonal lines for a given *idiag* value, corresponding to Line 7 in Fig. 2. In the implementation, *MMI* is equal to the number of discrete angles per octant.

The maximum relative flux error is computed in all the cells shown in Eq. (4). Like computing the iterative source, there is no dependence among the computations of relative flux error in all cells. The OpenMP primitive *parallel_for* is also applied to compute the maximum relative flux error in cells of some I-line grid columns in parallel. The reduction function directive *max* is conducted to get the maximum among the results of all the OpenMP threads:

$$Error_{\max} = max \mid \frac{\Phi(\vec{r})_i - \Phi(\vec{r})_{i-1}}{\Phi(\vec{r})_i} \mid . \qquad (4)$$

### B. Vector parallelism

In the thread-level parallelization, the granularity of distributing the tasks is one I-line grid column. Therefore, vector units can be used to exploit the parallelism in one I-line grid column, shown in Fig. 4. Except for computing relative flux error and solving recursive $S_n$ equations, all other procedures of Sweep3D are easily vectorized due to the independence among the calculations of all cells in one I-line grid column. Although the computations of relative flux error for all cells are also independent, it is related to the division operation and there is one branch, which is generated by the judgement of the zero value. Thus, computing relative flux error is processed by scalar units, rather than vector units in MIC. In the following, we mainly describe the implementation of exploiting vector parallelism in solving of recursive $S_n$ equations.

For the solving of recursive $S_n$ equations, the balanced equation, shown in Eq. (2), is transformed to Eq. (5) in one I-line grid column and the auxiliary equations are shown in Eq. (6).

$$\begin{aligned}\Phi_n(\vec{r},\Omega_m) = \frac{1.0}{D_i}[&Q_m + C_i \cdot \Phi_n(I_i,\Omega_m)^i \\ &+ C_j \cdot \Phi_n(J_i,\Omega_m)^i \\ &+ C_k \cdot \Phi_n(K_i,\Omega_m)^i],\end{aligned} \qquad (5)$$

$$\Phi_n(\vec{r},\Omega_m)^o_{I,J,K} = 2\Phi_n(\vec{r},\Omega_m) - \Phi_n(\vec{r},\Omega_m)^i_{I,J,K}, \qquad (6)$$

where $\Phi_n(I_i,\Omega_m)^i$, $\Phi_n(J_i,\Omega_m)^i$, and $\Phi_n(k_i,\Omega_m)^i$ are the input fluxes of the $(i,j,k)$ cell in the $I$, $J$, $K$ direction, respectively; $\Phi_n(\vec{r},\Omega_m)^o_{I,J,K}$ is the output flux in the $I$, $J$, $K$ direction of the $(i,j,k)$ cell, respectively; $\Phi_n(\vec{r},\Omega_m)$ is the central flux of the cell for the current discrete angle; $D_i$, $C_i$, $C_j$, and $C_k$ present the relative difference parameters. The central flux $\Phi_n(\vec{r},\Omega_m)$ can not be negative when the input fluxes $\Phi_n(\vec{r},\Omega_m)^i_{I,J,K}$ are non-negative in Eq. (5), while the out fluxes $\Phi_n(\vec{r},\Omega_m)^o_{I,J,K}$ obtained through Eq. (6) may be negative. We can choose whether the negative fluxes should be fixed up in the sub-procedure. If flux fixup is on, solving recursive $S_n$ equations would be full of judgements and branches like computing maximum flux error, and then is hard to be vectorized. If flux fixup is off, there is still data dependence in solving recursive $S_n$ equations in I-line grid columns. However, we can use loop unrolling and splitting to implement the parallelization of the sub-procedure.

The parallelization of the simplified $S_n$ recursion without flux fixup is shown in Algorithm 1. The arrays *B* and *C* stores $\Phi_n(\vec{r},\Omega_m)^{i,o}_{J,K}$. The array *phi* stores $\Phi_n(\vec{r},\Omega_m)$. *it* stands for the problem size of the I dimension and is determined by the geometry of the simulation. *phiir* stores the output and input flux in the I direction of the $(i,j,k)$ cell and is the dependence variable in the sub-procedure. Thus, we can isolate the computations (Lines 6–8) relating to *phiir*, and the left parts (Lines 3–5 and 9–11) for cells in one I-line grid column can be finished in a parallel way, respectively. The factor of



chinaXiv:202306.00232v1

ChinaXiv合作期刊



**Algorithm 1:** Parallelization of the simplified $S_n$ recursion without flux fixup

```
   input : Φ_n(r⃗, Ω_{m−1}), Φ_n(r⃗, Ω_m)^i_{I,J,K}
   output: Φ_n(r⃗, Ω_m), Φ_n(r⃗, Ω_m)^o_{I,J,K}
1  if !do_fixup then
       // Suppose that it could be divisible by
          8
2      for i=0; i < it; i= i +8 do
3          for l=0; l < 8; l++ do in parallel
4              phi[i + l] ←
               (1.0/D)(phi[i + l] + C_j · B[i + l] + C_k · C[i + l]))
5              C_i ← C_i/D
6          for l=0; l < 8; l++ do
7              phi[i + l] ← phi[i + l] + C_i · phiir
8              phiir = 2.0 · phi[i + l] − phiir
9          for l=0; l < 8; l++ do in parallel
10             B[i + l] = 2.0 · phi[i + l] − B[i + l]
11             C[i + l] = 2.0 · phi[i + l] − C[i + l]
```

loop unrolling is set to eight, so as to keep good data locality in the sub-procedure. Compared to serial implementation, only two memory store and one load operations are added, but most operations of the sub-procedure are processed in parallel. The implementation of Lines 9–11 in Algorithm 1 using vector intrinsic functions on MIC is shown in Lines 14–18 of Algorithm 2. Further, the software pre-fetch operations (Lines 8–9) are inserted to improve memory access cost. All the vectorized procedures of MIC-based Sweep3D in Fig. 4 are implemented by the use of intrinsic functions.

**Algorithm 2:** Vectorization of the simplified $S_n$ recursion without flux fixup using intrinsic functions

```
   input : Φ_n(r⃗, Ω_{m−1}), Φ_n(r⃗, Ω_m)^i_{I,J,K}
   output: Φ_n(r⃗, Ω_m), Φ_n(r⃗, Ω_m)^o_{I,J,K}
1  __m512d  v0, v1, v2, v3
2  if !do_fixup then
3      v0 = _mm512_set1_pd(2.0E + 00)
       // Suppose that it could be divisible by
          8
4      for i=0; i < it; i= i +8 do
5          ...
6          _v1 = _mm512_load_pd((void*)&B[i])
7          _v2 = _mm512_load_pd((void*)&C[i])
8          _mm_prefetch(&B[i + 16], _MM_HINT_T0)
9          _mm_prefetch(&C[i + 16], _MM_HINT_T0)
10         ...
11         for l=0; l < 8; l++ do
12             phi[i + l] ← phi[i + l] + C_i · phiir
13             phiir = 2.0 · phi[i + l] − phiir
14         _v3 = _mm512_load_pd((void*)&phi[i])
15         _v1 = _mm512_fmsub_pd(_v0, _v3, _v1)
16         _mm512_store_pd((void*)&B[i], _v1)
17         _v2 = _mm512_fmsub_pd(_v0, _v3, _v2)
18         _mm512_store_pd((void*)&C[i], _v2)
```

## IV. PERFORMANCE RESULTS AND DISCUSSION

### A. Experimental Setup

The experiment platform contains an eight-core Intel Xeon E5-2660 CPU with 128GB of memory, a MIC coprocessor, and a GPU coprocessor. The target MIC coprocessor includes 57 cores running the Intel MIC software stack. A C and Fortran hybrid version of Sweep3D is implemented on MIC and compiled by Intel C and Fortran compiler with level-three optimization. To compare MIC to GPU, the GPU version of Sweep3D in Ref. [10] is used and compiled by the nvcc compiler with level-three optimization. Both codes run in the double precision floating point. The specifications are described in Table 1.

Table 1. Specification of the experiment platform

| | |
|---|---|
| CPU | Intel Xeon E5-2660, 8 cores, 2.2 GHz |
| MIC | 57 cores, 4 threads/core, 1.1 GHZ, 5 G GDDR5 |
| GPU | Tesla K20C, 2496 CUDA Cores, 0.71 GHz, Capability 3.5, 5 G memory |
| Operating System | Linux Red Hat 4.4.5-6 |
| Intel Compiler | Intel v13.0.0, ifort, icc, OpenMP v3.1 |
| Nvcc Compiler | GCC v4.4.6, NVCC v6.0.1 |

In the simulation, we consider the vacuum boundary condition, which corresponds to a zero incoming flux in the domain, and DSA face-current calculations. In addition, the $P_n$ scattering order is set to 1, and center (1/3)-cubed grid points have a unit source in 3D geometry. The number of source iterations is set to four in Sweep3D. We run each instance of Sweep3D ten times and take the shortest runtime as the instance.

### B. Performance under different optimizations

In order to get the best performance, the affinity type between the OpenMP threads and hardware cores is set to the *scatter* type, and OpenMP threads can be allocated to as many hardware cores as possible in MIC in the performance evaluation.

The execution times of different optimizations without flux fixup are shown in Fig. 5. As the total memory usage increases with the number of spatial cells, the maximum problem size is set to $320^3$, in which a memory of 4201 MB is required. When the problem size is equal to $128^3$, the vectorization can make the execution time decreases by 29% in comparison with the implementation of only exploiting thread-level parallelism. When the problem size increases to $256^3$, the reduction of the runtime can reach up to 50%. When the maximum problem size is tested, a runtime reduction of about 44% is still obtained. Therefore, it's important for the parallelization of Sweep3D without flux fixup to do the vectorization on MIC. However, the performance improvement by vectorization is far less than the number of operations one vector unit can deal with simultaneously in MIC. The main reason is



chinaXiv:202306.00232v1

ChinaXiv合作期刊



that the performance mainly depends upon the memory performance of MIC due to the small ratio between floating point operations and memory loads in Sweep3D. In other words, the performance of the algorithm without flux fixup highly relies on the off-chip memory bandwidth in MIC.

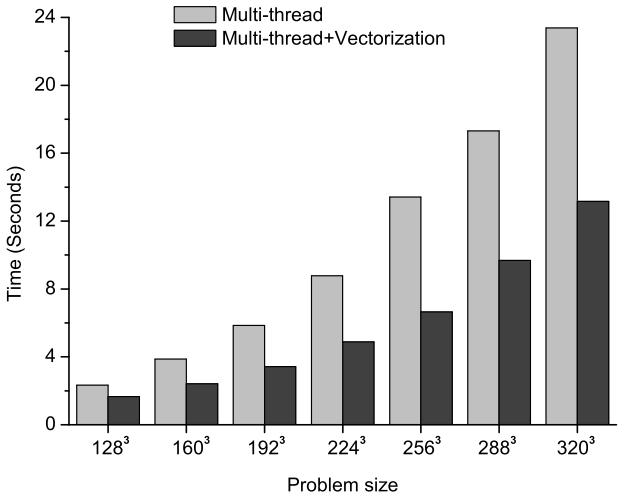


Fig. 5. Execution times of Sweep3D on MIC under different parallelizations and problem sizes when flux fixup is off. In the test, only 57 cores (228 threads) of the target MIC are used while the cores of the host CPU are not applied.

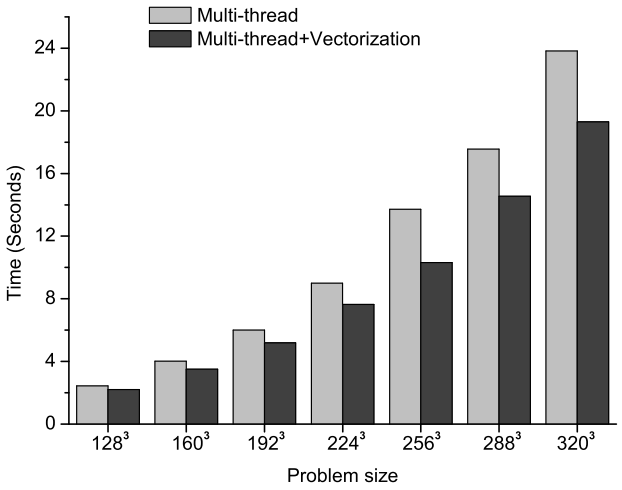


Fig. 6. Execution times of Sweep3D on MIC under different parallelizations and problem sizes when flux fixup is on. In the test, only 57 cores (228 threads) of the target MIC are used while the cores of the host CPU are not applied.

Figure 6 shows the performance comparisons of Sweep3D with the negative flux fixup between different optimizations. When the problem size is $128^3$, the vectorization optimization reduces the execution time only by 10%. When the problem size grows, the decrease in the runtime is at most 25%. Compared to the runtime reduction by vectorization without flux fixup, the decrease in the runtime with flux fixup is much smaller. There are two main reasons for this phenomenon. One reason is that the sub-procedure of solving recursive $S_n$ equations with flux fixup is full of judgement and hard to be processed with vector units. The other reason is the proportion of the runtime in solving recursive $S_n$ equations in the total execution time with flux fixup is higher than that without flux fixup. Therefore, the exploitation of thread parallelism rather than vector parallelism is the key to get a high performance from the discrete ordinates method with flux fixup on MIC.

To assess the absolute performance of different optimizations, we apply the Roofline model [19] to the target MIC (Fig. 7). The Roofline model can measure the maximum performance an algorithm can achieve on a given hardware architecture. The Roofline model can compare the performance of a given algorithm on different systems, as well. There are three basic factors in the Roofline model, including the peak floating-point performance (GFlops/s), the peak memory bandwidth (GBytes/s), and the operational intensity (Flops/Byte). The peak performance and the peak memory bandwidth rely on the hardware systems. Two horizontal lines in Fig. 7 show the peak FLOPS under different optimizations, which are found through the hardware specifications of the target MIC. The peak memory bandwidth is measured with the STREAM benchmark [20]. Using the STREAM-BIG-C test, we obtain the peak memory bandwidth of 80.62 Gbytes/s. The slanted line in Fig. 7 exhibits the maximum performance an algorithm can reach, owing to the limit of the memory bandwidth. The Operational Intensity (OI) means the number of FLOPS per byte of traffic between the caches and memory, and is determined by the algorithms. In Sweep3D, the wavefront sweeping subroutine often takes more than 90% of the total runtime. Thus, we only consider the wavefront sweeping subroutine in the comparison between the theoretical performance and the experimental performance. Without flux fixup, the operational intensity of the subroutine is constant for any problem size and any source iteration step. With flux fixup, the operational intensity increases with the number of flux fixup in the subroutine. Therefore there are different operational intensities in different source iterations.

The comparison between theoretical performance and experimental performance under different optimizations is shown in Table 2. Without flux fixup, the operational intensity of the wavefront sweeping procedure is about 0.3583 Flops/Byte. With flux fixup, the minimum and maximum operational intensities in the four iterations are 0.3616 Flops/Byte and 0.3811 Flops/Byte respectively. As seen from Fig. 7, the theoretical performance is given by the product of the memory bandwidth and the operational intensity due to the low operational intensities. The experimental results are obtained by measuring the execution time of the wavefront sweeping subroutine of the MIC-based Sweep3D. When flux fixup is off, the theoretical performance is 28.89 Gflops/s and the maximum experimental performance is 21.04 Gflops/s. When flux fixup is on, the theo-

chinaXiv:202306.00232v1

ChinaXiv合作期刊



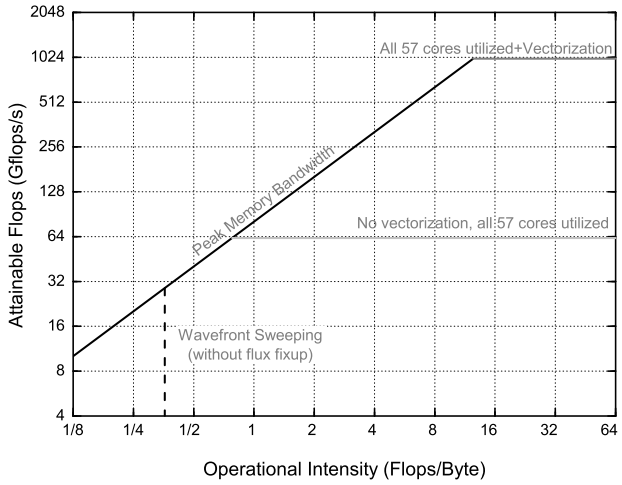


Fig. 7. Roofline model with ceilings for the wavefront sweeping subroutine of Sweep3D on the target MIC. The graph is on a log-log scale.

retical performance increases with operational intensity. The experimental performance with the maximum operational intensity is less than that with the minimum operational intensity under the same optimizations. The chief reason is that the fixup of flux leads to worsen the load balance between the OpenMP threads. With flux fixup, the minimum theoretical performance is 29.15 Glops/s and the corresponding maximum performance is 14.94 Gflops/s. Altogether, all the experimental performances are lower than the theoretical performances, and the maximum efficiency is about 72.8%. Two factors incur the gap. One factor is that the varying thread parallelism along the sweep directions leads to load imbalance among hardware threads and the exploited vector parallelism is insufficient for filling up the SIMD pipeline of each core in MIC. The other is the imperfection of the Roofline model, which is just insightful. Actually, the obtainable peak memory bandwidth is influenced by the number of threads which relies on the varying thread parallelism. Also, the synchronization and scheduling overhead of many OpenMP threads can't be omitted and rises fast with the number of OpenMP threads. But neither the variance of memory bandwidth nor the overhead of multi-threading technology is included in the Roofline model, so that the obtained theoretical performance is high by using it.

Table 2. Comparison between prediction performance and experimental performance of the wavefront sweeping subroutine of Sweep3D on MIC under different optimizations (the problem size is $320^3$, the unit of performance is Gflops/s)

| Flux fixup | Off | | On | | | |
|---|---|---|---|---|---|---|
| OI (Flops/Byte) | 0.3583 | | 0.3616 (Min) | | 0.3811 (Max) | |
| Performance | Pred. | Expt. | Pred. | Expt. | Pred. | Expt. |
| Multi-thread | 28.89 | 11.96 | 29.15 | 12.47 | 30.72 | 11.60 |
| Multi-thread + Vectorization | 28.89 | 21.04 | 29.15 | 14.94 | 30.72 | 13.35 |

### C. Performance comparison between MIC and GPU

Compared with GPU, the speedup of Sweep3D on MIC under various problem sizes is shown in Fig. 8. The double precision floating-point peak performance ratio of the target MIC to GPU is about 0.85 : 1.00. The speedup gradually increases with the problem sizes, whether the flux fixup is on or not.

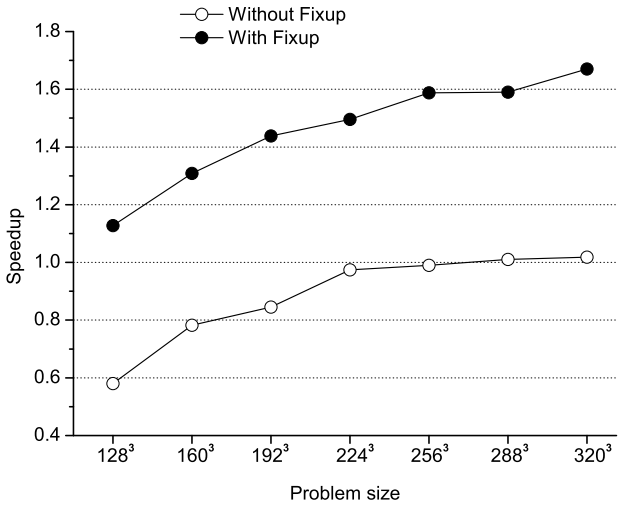


Fig. 8. Speedup of Sweep3D on MIC compared to that on GPU under various problem sizes.

Without flux fixup, MIC is inferior to GPU when the problem size is less than $288^3$. When the problem size is $128^3$, the performance ratio of MIC to GPU is only 0.58 : 1.00. Only when the problem size increases to $192^3$, is the performance ratio roughly equal to the peak performance ratio. This phenomenon is mainly caused by the insufficient utilization of cores. The hardware cores in MIC act like the Streaming Multiprocessors (SMs) in GPU. There are 57 cores in the target MIC, while there are only 13 SMs in the target GPU. Therefore, the utilization of cores in MIC is worse than that in GPU with the given small problem sizes. When the problem size is greater than or equal to $288^3$, MIC is as good as GPU.

When the flux fixup is on, MIC is superior to GPU with all examined problem sizes. When the smallest problem size $128^3$ is tested, the performance speedup between MIC and GPU is 1.13 times. When the biggest problem size $320^3$ is studied, a speedup of up to 1.67 times is procured. The GPU architecture consists of simple cores, while the MIC architecture is based on general-purpose cores. Thus MIC is much better at processing the judgement expressions than GPU. As stated in Sec. IV B, solving recursive $S_n$ equations with flux fixup includes many judgement expressions and is hard to be efficiently parallelized with vector units in MIC as well as SIMT units in GPU. Therefore, we can get much better performance on MIC than GPU with flux fixup.

chinaXiv:202306.00232v1

ChinaXiv合作期刊



chinaXiv:202306.00232v1

### D. Scalability with the number of cores

In order to effectively evaluate the variance of the performance with the number of cores, the OpenMP threads are bound to specific hardware cores in the scalability evaluation. There are two primary ways to scale Sweep3D on MIC, including strong scaling and week scaling. Strong scaling means that more cores are applied to the same problem size to get results faster. Weak scaling refers to the concept of increasing the problem size as Sweep3D runs on more cores.

#### 1. *Strong scaling evaluation*

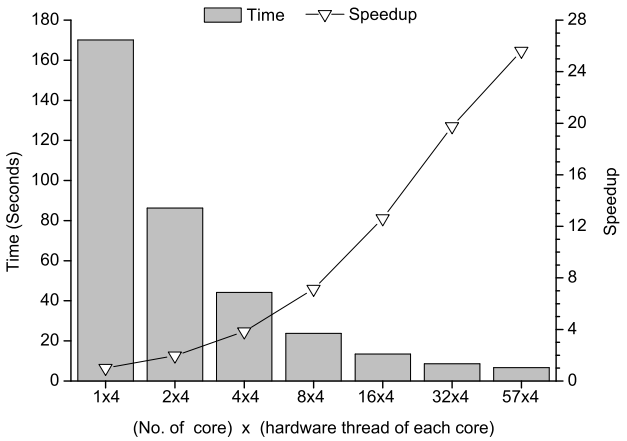


Fig. 9. Execution times of Sweep3D running on different number of cores in MIC and speedup in comparison with the simulation on only one core of MIC when flux fixup is off. The problem size is $256^3$.

Figure 9 shows the execution times of Sweep3D without flux fixup running on different number of cores in MIC when the problem size is $256^3$. The horizontal ordinate represents the number of hardware threads which equals that of the cores times that of hardware threads per core. One core consists of four hardware threads in MIC. In order to take full advantage of those four hardware threads, four OpenMP threads need to be assigned to one core. Using all 57 cores in the target MIC, there are 228 parallel hardware threads and, therefore, the number of OpenMP threads are required to be 228. When only one core is used, the runtime of Sweep3D is 170.08 seconds. Two cores make the runtime reduce to 86.23 seconds. The speedup of performance from 1 to 2 cores is about 1.97 times. When the number of cores increases to 32, the runtime drops to 8.61 seconds and the speedup from 1 to 32 cores reaches 19.76 times. When all cores in the target MIC are utilized, the runtime is 6.64 s and a 25.6-fold speedup is achieved from 1 to 57 cores. Therefore, we can infer that the runtime would diminish continually with more cores.

Figure 10 shows the performance of Sweep3D with a negative flux fixup running on different number of cores in MIC when the problem size is $256^3$. As stated in Sec. IV B, the key

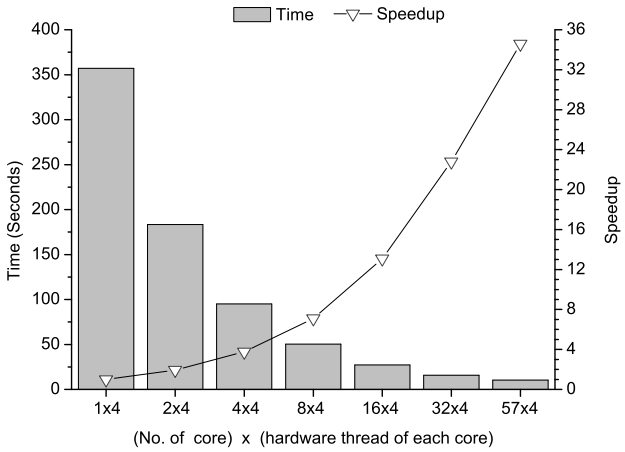


Fig. 10. Execution times of Sweep3D running on different number of cores in MIC and speedup in comparison with the simulation on only one core of MIC when flux fixup is on. The problem size is $256^3$.

to achieving high performance with Sweep3D with flux fixup on MIC lies in the exploitation of thread parallelism, rather than vector parallelism. Therefore, a much bigger speedup can be reached when more cores or hardware threads are employed. The speedup of performance from 1 to 32 cores is about 22.78 times, while it reaches 34.59 times from 1 to 57 cores.

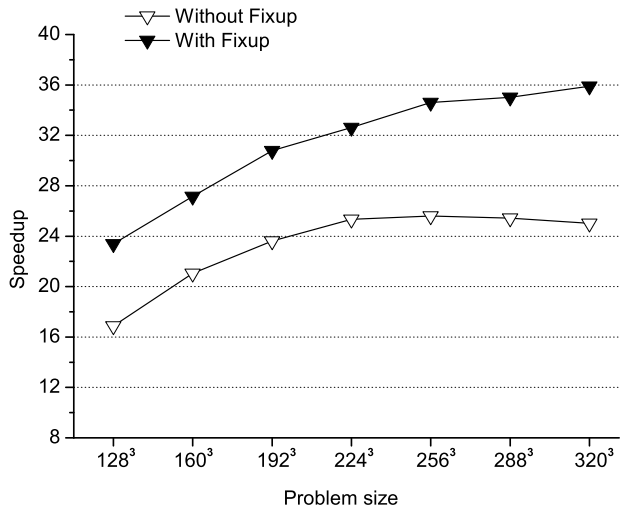


Fig. 11. Speedup of Sweep3D running on all 57 cores of MIC in comparison with that on only 1 core under different problem sizes.

Figure 11 exhibits the speedup of performance from 1 to 57 cores when Sweep3D runs under different problem sizes. Without flux fixup, the speedup rises gradually with problem size at the beginning. As the problem size is equal to $256^3$, the maximum speedup of 25.61 times is obtained. But under larger problem sizes, a plateau is reached and even tiny drops can be observed. This is because the simulation without flux

ChinaXiv合作期刊



fixup is limited by the peak memory bandwidth and so more parallelism with larger problem sizes can't be efficiently exploited by parallel cores. Hence, the maximum strong scaling efficiency is about 45% without flux fixup. With flux fixup, the speedup increases continuously with problem size. When the problem size reaches $320^3$, the speedup from 1 to 57 cores achieves 35.90 times. Therefore, the maximum strong scaling efficiency can reach 63% with flux fixup. However, the strong scaling efficiency is not high in all cases. The runtime is chiefly determined by wavefront sweeping in Sweep3D. The number of parallel I-lines in wavefront sweeping first increases from one and then decreases to one with particle flowing, shown in Fig. 1. Thus there are load imbalance among cores when more than one core is utilized. The application of more cores will further aggravate the load imbalance and then the utilization efficiency of the cores reduces with the number of cores. As a result, the obtained strong scaling efficiency is not high.

#### 2. *Weak Scaling Evaluation*

Table 3 shows the execution times of Sweep3D running on different number of cores in MIC. For the problem sizes, the sizes of the I and J dimensions are both fixed to 256, and the size of the K dimension keeps a linear relationship with the number of utilized cores in MIC. When the number of cores is less than or equal to 16, the runtime exhibits small variations with the problem sizes. When more than 16 cores are used, the runtime changes greatly with the problem size. A chief cause is also the load imbalance that the changing parallelism incurs in the sweeping. The load imbalance sharply worsens under more than 16 cores and the corresponding problem size, and therefore the runtime increases rapidly. Without flux fixup, the runtime under only one core is 7.25 seconds and rises to 11.59 seconds finally. Hence the weak scaling efficiency is about 63% when flux fixup is off. With flux fixup, the execution time is 14.66 seconds at the beginning and grows to 17.59 seconds at the end. Thus, a weak scaling efficiency of 83% is reached when flux fixup is on.

Table 3. Execution times of Sweep3D when the problem sizes increase linearly with the number of utilized cores in MIC

| No. of core × hardware thread/core | Problem size | Runtime (s) | |
|---|---|---|---|
| | | Without flux fixup | With flux fixup |
| 1 × 4 | 256×256× 8 | 7.25 | 14.66 |
| 2 × 4 | 256×256× 16 | 7.08 | 14.60 |
| 4 × 4 | 256×256× 32 | 7.28 | 14.27 |
| 8 × 4 | 256×256× 64 | 7.23 | 14.33 |
| 16 × 4 | 256×256× 128 | 7.75 | 14.90 |
| 32 × 4 | 256×256× 256 | 8.61 | 15.67 |
| 57 × 4 | 256×256× 456 | 11.59 | 17.59 |

Overall, the results above show that our implementation has good scalability on MIC, regardless of whether fix fixup is on, and can also achieve good performance on MIC with more cores.

### E. Compare with previous work

The difference between the new implementation in this paper and our old one [13] is that the software pre-fetch technology is utilized to improve the memory access cost in the vectorization of our new implementation. For the problem size $256^3$, the new implementation and the old one cost 6.64 and 6.78 seconds, respectively, disabling flux fixup and using all 57 cores in MIC. When only one core in MIC is utilized, the execution times are 170.08 and 187.37 seconds, respectively. With flux fixup, the new costs are 10.32 seconds on all cores and 356.98 seconds on one core. Correspondingly, the old demands 10.41 and 370.49 seconds. So the minimum and maximum performance improvements of our new implementation based on the old one are 0.93% and 10.17% respectively. Thereby, our new implementation shows better performance portability with the number of cores in MIC than the old one.

## V. CONCLUSION AND FUTURE WORK

In this paper, the parallel programming model, OpenMP, and vector intrinsic functions are used to implement the parallelization of Sweep3D on MIC. All the three procedures of Sweep3D are processed in parallel. Except for computing relative flux error and solving recursive $S_n$ equations with flux fixup, all other sub-procedures of Sweep3D are vectorized on MIC. We evaluate the influence of different parallel optimizations to the performance and apply the Roofline model to access the absolute performance of the optimizations. The results show that the performance of the algorithm is determined by the number of cores and the off-chip memory bandwidth in MIC without flux fixup, while the speed of the algorithm is mainly dependent on the number of cores in MIC with flux fixup. The scalability of the MIC-based Sweep3D with the number of cores is investigated. The investigation demonstrates that the MIC-based implementation has good scalability in performance with the Intel MIC architecture. Moreover, the comparison between our implementation on MIC and the prior running on Tesla K20C GPU is also discussed.

In the future, the performance and scalability issues of Sweep3D on heterogeneous CPU/MIC clusters like Tianhe-2 supercomputer will be studied.

[1] Colomer G, Borrell R, Trias F X, *et al*. Rodrguez, parallel algorithms for $S_n$ transport sweeps on unstructured meshes. J Comput Phys, 2013, **232**: 118–135. DOI:



chinaXiv:202306.00232v1

ChinaXiv合作期刊



10.1016/j.jcp.2012.07.009
[2] Marchuk G I and Lebedev V I. Numerical methods in the theory of neutron transport. New York (USA): Harwood Academic Publishers, 1986, 3–26.
[3] Hoisie A, Lubeck O, Wasserman H, *et al*. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. Int J High Perform C, 2000, **14**: 330–346. DOI: 10.1177/109434200001400405
[4] Fischer J W and Azmy Y Y. Comparison via parallel performance models of angular and spatial domain decompositions for solving neutral particle transport problems. Prog Nucl Energ, 2007, **49**: 37–60. DOI: 10.1016/j.pnucene.2006.08.003
[5] Hoisie A, Lubeck O and Wasserman H. Scalability analysis of multidimensional wavefront algorithms on large-scale SMP clusters. 7th Symposium on the Frontiers of Massively Parallel Computation, 1999, 4–15. DOI: 10.1109/FMPC.1999.750452
[6] Los Alamos National Laboratory. The ASCI Sweep3D Benchmark. http://wwwc3.lanl.gov/pal/software/sweep3d/
[7] Petrini F, Fossum G, Fernandez J, *et al*. Multicore surprises: lessons learned from optimizing Sweep3D on the cell broadband engine. IEEE International Parallel and Distributed Processing Symposium, 2007, 1–10. DOI: 10.1109/IPDPS.2007.370252
[8] Lubeck O, Lang M, Srinivasan R, *et al*. Implementation and performance modeling of deterministic particle transport (Sweep3D) on the IBM Cell/B.E.. Sci Program, 2009, **17**: 199–208. DOI: 10.3233/SPR-2009-0266
[9] Gong C, Liu J, Gong Z, *et al*. Optimizing sweep3d for graphic processor unit. Lect Notes Comput Sc. 2010, **6081**: 416–426. DOI: 10.1007/978-3-642-13119-6_36
[10] Gong C, Liu J, Chi L, *et al*. GPU accelerated simulations of 3D deterministic particle transport using discrete ordinates method. J Comput Phys, 2011, **230**: 6010–6022. DOI: 10.1016/j.jcp.2011.04.010
[11] Davis K, Hoisie A, Johnson G, *et al*. A performance and scalability analysis of the BlueGene/L architecture. Proceedings of the ACM/IEEE SC2004 Conference on Supercomputing, IEEE Press, 2004, 41. DOI: 10.1109/SC.2004.8
[12] Barker K J, Davis K, Hoisie A, *et al*. Entering the petaflop era: The architecture and performance of Roadrunner. Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, 2008, 1–11. DOI: 10.1109/SC.2008.5217926
[13] Wang Q, Xing Z, Liu J, *et al*. Parallel 3D deterministic particle transport on Intel MIC architecture. Proceedings of 2014 International Conference on High Performance Computing and Simulation (HPCS), IIEEE Press, 2014, 186–192. DOI: 10.1109/HPCSim.2014.6903685
[14] Intel Corporation. Knights corner software developers guide revision 1.03, 2012, 11-35.
[15] Anthony S. Intel unveils 72-core x86 knights landing CPU for exascale supercomputing. http://www.extremetech.com/extreme/
[16] Wang Q, Liu J, Tang X, *et al*. Accelerating embarrassingly parallel algorithm on Intel MIC. Proceedings of 2014 International Conference on Progress in Informatics and Computing, IEEE Press, 2014, 213–218. DOI: 10.1109/PIC.2014.6972327
[17] Saule E and Catalyurek U V. An Early Evaluation of the Scalability of Graph Algorithms on the Intel MIC Architecture. In: Proceedings of 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, 2012, **18**: 1629–1639. DOI: 10.1109/IPDPSW.2012.204
[18] Adams M L and Larsen E W. Fast iterative methods for discrete-ordinates particle transport calculations. Prog Nucl Energ, 2002, **40**: 3–159. DOI: 10.1016/S0149-1970(01)00023-3
[19] Williams S, Waterman A and Patterson D. Roofline: an insightful visual performance model for multicore architectures. Commun Acm, 2009, **52**: 65–76. DOI: 10.1145/1498765.1498785
[20] McCalpin J D. STREAM: Sustainable memory bandwidth in high performance computers, 1995. http://www.cs.virginia.edu/stream/

chinaXiv:202306.00232v1